Practical 7: Unsupervised Hebbian learning and constraints

Neural Computation 2004-05
Mark van Rossum
2nd March 2005

Comment: WITH ANSWERS
In this practical we discuss different normalisation methods used for Hebbian learning and how they determine the final weights. We consider a single linear output neuron.

Input correlations
First, we need to create input data. For simplicity we have just 2 inputs. The inputs can be uncorrelated, positive correlated, or negatively correlated. The script below shows a simple way to create correlated data.

In case of visual input, try to think of cases in which inputs which are uncorrelated, correlated or negatively correlated.

Unconstrained learning
First, we implement a plain Hebb rule. Pick an initial value for the synaptic weights. Present the input data drawn from the correlated distribution (some 100 trials). Update with every presentation the weights according to the Hebb rule. Try this for a variety of initial weights. You can make a vector field plot using the quiver command. In order to prevent run away learning, impose maximal and minimal weights. For instance, $0 < w < 1$ or $-1 < w < 1$.

Try for positively and negatively correlated inputs. What is the steady state result of this plain Hebb rule?
Comment: Result: When anti-correlated, see fig. 12.3 of notes. When $-1 < w < 1$, endpoints are 1,1 and (-1,1).
When the inputs are positively correlated, picture is 90deg. rotated. (so sym. breaking depends on choice of limits).

Constrained learning
Next, we implement options to include multiplicative or subtractive scaling. Both constraints modify the learning rule such that the sum of weights is constant. (see script). Show that the sum of weights is indeed constant under these modified learning rules.

Simulate this. How is this reflected in the vector field plots? How do the constraints lead to competition? Which other constraints than keeping the sum constant would be possible?
Consider the development of monocular dominance columns in V1. One can assume that the input from both eyes is strongly correlated. Which type of constrained learning would explain such development? Consider a population of output neurons, how would their fate be determined? Could lateral connectivity change the conclusion about the required learning rule?

Comment: Result:
- Subtractive; uncorrelated, or pos. or neg. correlated always \rightarrow (1,0) (0,1). Note, it can not move from the line \( x+y=c \).
- Mult: pos. (or uncorr) toward \( w_1=w_2 \) diagonal. Neg. corr: away from diagonal. Lateral inhibition can break symmetry.

Oja’s rule

Implement Oja’s rule. What is the difference with the previous approaches?

Comment: Result: no hard bounds are needed.

Extras: You can use this layout to do BCM. Bit tricky.

Matlab script

```matlab
%hebb
close all;
negcor = 1; % {0,1} pos. or neg. input correlation
subnorm = 0; % {0,1} use subtractive normalisation?
multnorm = 1; % {0,1} use multiplicative norm?
wmax = 1; % hard limits on weight
wmin = 0;
ntr=100;
inmat = zeros(2,ntr);
ranx = randn(1,ntr); rany = randn(1,ntr); ranz = randn(1,ntr);
inmat(1,:)=(ranx+(1-2*negcor)*ranz)/sqrt(2);
inmat(2,:)=(rany+ranz)/sqrt(2);
plot(inmat(1,:),inmat(2,:),'x')
cov(inmat') %note matlab has function for coVARiance matrix, not for correlation matrix

eta=0.01/ntr;% learning rate
%vectorplot
qx=[]; qy=[]; qu=[]; qv=[];
% pick starting values for weight
for w1init=0:0.1:1
w1 = w1init;
...
for itr=1:ntr
x1 = inmat(1,itr);
x2 = inmat(2,itr);
y = w1*x1+w2*x2;
...
...  dw1 = dw1 -y*(x1+x2)/2; % subtractive normalisation
...
dw1 = dw1 -y*(x1+x2)/(w1+w2)*w1; % multiplicative
...
w1 = w1+ eta*dw1;
```
% impose hard limits to maximal weight wmin < w < wmax
w1 = max(wmin, min(wmax, w1));
end % itr-loop
qx = [qx w1init];
qy = [qy w2init];
qu = [qu w1-w1init];
qv = [qv w2-w2init];
end % w1init-loop
figure
quiver(qx,qy,qu,qv)